format manual page - Tcl Built-In Commands

tcl.tk/man/tcl/TclCmd/format.htm

NAME

format — Format a string in the style of sprintf

SYNOPSIS

format *formatString* ?*arg arg ...*?

INTRODUCTION

This command generates a formatted string in a fashion similar to the ANSI C **sprintf** procedure. *FormatString* indicates how to format the result, using **%** conversion specifiers as in **sprintf**, and the additional arguments, if any, provide values to be substituted into the result. The return value from **format** is the formatted string.

DETAILS ON FORMATTING

The command operates by scanning *formatString* from left to right. Each character from the format string is appended to the result string unless it is a percent sign. If the character is a **%** then it is not copied to the result string. Instead, the characters following the **%** character are treated as a conversion specifier. The conversion specifier controls the conversion of the next successive *arg* to a particular format and the result is appended to the result string in place of the conversion specifier. If there are multiple conversion specifiers in the format string, then each one controls the conversion of one additional *arg*. The **format** command must be given enough *arg*s to meet the needs of all of the conversion specifiers in *formatString*.
Each conversion specifier may contain up to six different parts: an XPG3 position specifier, a set of flags, a minimum field width, a precision, a size modifier, and a conversion character. Any of these fields may be omitted except for the conversion character. The fields that are present must appear in the order given above. The paragraphs below discuss each of these fields in turn.

OPTIONAL POSITIONAL SPECIFIER

If the **%** is followed by a decimal number and a **$**, as in "**%2$d**", then the value to convert is not taken from the next sequential argument. Instead, it is taken from the argument indicated by the number, where 1 corresponds to the first *arg*. If the conversion specifier requires multiple arguments because of ***** characters in the specifier then successive arguments are used, starting with the argument given by the number. This follows the XPG3 conventions for positional specifiers. If there are any positional specifiers in *formatString* then all of the specifiers must be positional.

OPTIONAL FLAGS

The second portion of a conversion specifier may contain any of the following flag characters, in any order:

-
Specifies that the converted argument should be left-justified in its field (numbers are normally right-justified with leading spaces if needed).

+
Specifies that a number should always be printed with a sign, even if positive.

space
Specifies that a space should be added to the beginning of the number if the first character is not a sign.

0
Specifies that the number should be padded on the left with zeroes instead of spaces.

#
Requests an alternate output form. For **o** conversions it guarantees that the first digit is always **0**. For **x** or **X** conversions, **0x** or **0X** (respectively) will be added to the beginning of the result unless it is zero. For **b** conversions, **0b** will be added to the beginning of the result unless it is zero. For all floating-point conversions (**e**, **E**, **f**, **g**, and **G**) it guarantees that the result always has a decimal point. For **g** and **G** conversions it specifies that trailing zeroes should not be removed.

OPTIONAL FIELD WIDTH

The third portion of a conversion specifier is a decimal number giving a minimum field width for this conversion. It is typically used to make columns line up in tabular printouts. If the converted argument contains fewer characters than the minimum field width then it will be padded so that it is as wide as the minimum field width. Padding normally occurs by adding extra spaces on the left of the converted argument, but the **0** and **-** flags may be used to specify padding with zeroes on the left or with spaces on the right, respectively. If the minimum field width is specified as ***** rather than a number, then the next argument to the **format** command determines the minimum field width; it must be an integer value.

OPTIONAL PRECISION/BOUND

The fourth portion of a conversion specifier is a precision, which consists of a period followed by a number. The number is used in different ways for different conversions. For **e**, **E**, and **f** conversions it specifies the number of digits to appear to the right of the decimal point. For **g** and **G** conversions it specifies the total number of digits to appear, including those on both sides of the decimal point (however, trailing zeroes after the decimal point will still be omitted unless the **#** flag has been specified). For integer conversions, it specifies a minimum number of digits to print (leading zeroes will be added if necessary). For **s** conversions it specifies the maximum number of characters to be printed; if the string is longer than this then the trailing characters will be dropped. If the precision is specified with ***** rather than a number then the next argument to the **format** command determines the precision; it must be a numeric string.

OPTIONAL SIZE MODIFIER

The fifth part of a conversion specifier is a size modifier, which must be **ll**, **h**, or **l**. If it is **ll** it specifies that an integer value is taken without truncation for conversion to a formatted substring. If it is **h** it specifies that an integer value is truncated to a 16-bit range before converting. This option is rarely useful. If it is **l** it specifies that the integer value is truncated to the same range as that produced by the **wide()** function of the **expr** command (at least a 64-bit range). If neither **h** nor **l** are present, the integer value is truncated to the same range as that produced by the **int()** function of the **expr** command (at least a 32-bit range, but determined by the value of the **wordSize** element of the **tcl_platform** array).

MANDATORY CONVERSION TYPE

The last thing in a conversion specifier is an alphabetic character that determines what kind of conversion to perform. The following conversion characters are currently supported:

d
Convert integer to signed decimal string.

u
Convert integer to unsigned decimal string.

i
Convert integer to signed decimal string (equivalent to **d**).

o
Convert integer to unsigned octal string.

x or X
Convert integer to unsigned hexadecimal string, using digits “0123456789abcdef” for **x** and “0123456789ABCDEF” for **X**).

b
Convert integer to unsigned binary string, using digits 0 and 1.

c
Convert integer to the Unicode character it represents.

s
No conversion; just insert string.

f
Convert number to signed decimal string of the form *xx.yyy*, where the number of *y*'s is determined by the precision (default: 6). If the precision is 0 then no decimal point is output.

e or E
Convert number to scientific notation in the form *x.yyy***e±***zz*, where the number of *y*'s is determined by the precision (default: 6). If the precision is 0 then no decimal point is output. If the **E** form is used then **E** is printed instead of **e**.

g or G
If the exponent is less than -4 or greater than or equal to the precision, then convert number as for **%e** or **%E**. Otherwise convert as for **%f**. Trailing zeroes and a trailing decimal point are omitted.

%
No conversion: just insert **%**.

DIFFERENCES FROM ANSI SPRINTF

The behavior of the format command is the same as the ANSI C **sprintf** procedure except for the following differences:

1. Tcl guarantees that it will be working with UNICODE characters.
2. **%p** and **%n** specifiers are not supported.
3. For **%c** conversions the argument must be an integer value, which will then be converted to the corresponding character value.
4. The size modifiers are ignored when formatting floating-point values. The **ll** modifier has no **sprintf** counterpart. The **b** specifier has no **sprintf** counterpart.

EXAMPLES

Convert the numeric value of a UNICODE character to the character itself:

```
set value 120
set char [format %c $value]
```

Convert the output of **time** into seconds to an accuracy of hundredths of a second:

```
set us [lindex [time $someTclCode] 0]
puts [format "%.2f seconds to execute" [expr {$us / 1e6}]]
```

Create a packed X11 literal color specification:

```
# Each color-component should be in range (0..255)
set color [format "#%02x%02x%02x" $r $g $b]
```

Use XPG3 format codes to allow reordering of fields (a technique that is often used in localized message catalogs; see **msgcat**) without reordering the data values passed to **format**:

```
set fmt1 "Today, %d shares in %s were bought at $%.2f each"
puts [format $fmt1 123 "Global BigCorp" 19.37]

set fmt2 "Bought %2\$s equity ($%3$.2f x %1\$d) today"
puts [format $fmt2 123 "Global BigCorp" 19.37]
```

Print a small table of powers of three:

```
# Set up the column widths
set w1 5
set w2 10

# Make a nice header (with separator) for the table first
set sep +-[string repeat - $w1]-+-[string repeat - $w2]-+
puts $sep
puts [format "| %-*s | %-*s |" $w1 "Index" $w2 "Power"]
puts $sep

# Print the contents of the table
set p 1
for {set i 0} {$i<=20} {incr i} {
    puts [format "| %*d | %*ld |" $w1 $i $w2 $p]
    set p [expr {wide($p) * 3}]
}

# Finish off by printing the separator again
puts $sep
```